Scalable task distribution with Scala, Akka and Mesos

Dario Rexin
@evonox
What is Mesos?
What is Mesos?

- Apache open source project
- Distributed systems kernel
- Multi resource scheduler (CPU, Memory, Ports, Disk)
- Scalable to 10,000s of nodes
- Fault tolerant
- First class Docker support
Distributed Systems Kernel

- Runs on every node
- Aggregates all resources in the cluster
- Provides applications with APIs for resource management and scheduling
- Offers resources to applications in a fair (configurable) manner
Fault Tolerance
Why should I bother?

- Running many tasks on many machines
- Scaling up and down the number of tasks and machines
- Handling failures in the cluster
- Better resource utilization through multi-tenancy
Static partitioning Waste of resources!
Virtualization Operational overhead!
Multi tenancy + automatic task distribution
Multi tenancy + automatic task distribution
Say hi to Marathon
What is Marathon?

- Distributed init system
- Fault tolerant
- Manages deployments
- Checks health of applications
- Manages task and machine failures
- Runs Docker containers
Deploying apps with Marathon

curl -XPOST -H "Content-Type: application/json" http://marathon:8080/v2/apps -d '{
  "id": "my-app",
  "cmd": "python -m SimpleHTTPServer $PORT0",
  "cpus": 1,
  "mem": 64,
  "instances": 10,
  "ports": [0]
}'
Deploying apps with Marathon

```json
{
    "steps": [
        {
            "actions": [
                {
                    "action": "StartApplication",
                    "app": "/my-app"
                }
            ]
        }
    ]
}
```
Deploying apps with Marathon

<table>
<thead>
<tr>
<th>ID</th>
<th>Memory (MB)</th>
<th>CPUs</th>
<th>Tasks / Instances</th>
<th>Health</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>/my-app</td>
<td>64</td>
<td>1</td>
<td>0 / 10</td>
<td></td>
<td>Deploying</td>
</tr>
</tbody>
</table>
Deploying apps with Marathon

```
> {} =
{|id: /pyserv, cmd: "python -m SimpleHTTPServer $PORT0", args: null, user: null, env: {}, instances: 1, cpus: 1.0, mem: 128.0, disk: 0.0, executor: "", constraints: [], urls: [], storeUrls: [], ports: [0], requirePorts: false, backoffFactor: 1.15, container: null, healthChecks: [{"path": "/", "protocol": "HTTP", "portIndex": 0, "command": null, "gracePeriodSeconds": 300, "intervalSeconds": 0, "timeoutSeconds": 20, "maxConsecutiveFailures": 3}], dependencies: [], upgradeStrategy: {"minimumHealthyCapacity": 1.0, "maximumOverCapacity": 1.0}, labels: {}, version: "2015-03-06T01:49:45.601Z", tasks: [], deployments: [{"id": "Se24de0-8b12-4311-83f3-bb899ace6e6b", "tasksRunning": 0, "tasksHealthy": 0, "tasksUnhealthy": 0, "backoffSeconds": 1, "maxLaunchDelaySeconds": 3600}]}
```

```bash
  "id": "pyserv",
  "cmd": "python -m SimpleHTTPServer $PORT0",
  "ports": [0],
  "healthChecks": []
}' > /projects/erlelevator/bin (git master 9f2265a)
```
Deploying apps with Marathon
Deploying apps with Marathon

Scheduler Actor notifies Deployment Manager
Deploying apps with Marathon

Scheduler Actor \(\xrightarrow{\text{notifies}}\) Deployment Manager \(\xrightarrow{\text{starts}}\) Deployment Actor
Deploying apps with Marathon

- Scheduler Actor
- Deployment Manager
- Deployment Actor
- AppStart Actor

Scheduler Actor notifies Deployment Manager, which starts the Deployment Actor. The Deployment Actor also starts the AppStart Actor.
Deploying apps with Marathon

Scheduler Actor → notifies → Deployment Manager → starts → Deployment Actor → starts → AppStart Actor

EventStream
Deploying groups with Marathon

curl -XPOST -H "Content-Type: application/json" http://marathon:8080/v2/groups -d '{
    "id": "my-group",
    "apps": [
        {"id": "my-app", ... },
        {"id": "my-other-app", ... },
        {"id": "yet-another-app", ... }
    ]
}'
Deploying apps with Marathon

```json
{
  "steps": [
    {
      "actions": [
        { "action": "StartApplication", "app": "/my-app" },
        { "action": "StartApplication", "app": "/my-other-app" },
        { "action": "StartApplication", "app": "/yet-another-app" }
      ]
    }
  ]
}
```
Deploying groups with Marathon

Scheduler Actor \(\xrightarrow{\text{notifies}}\) Deployment Manager \(\xrightarrow{\text{starts}}\) Deployment Actor \(\xrightarrow{\text{starts}}\) AppStart Actor

EventStream
Dependencies between apps

curl -XPOST -H "Content-Type: application/json" http://marathon:8080/v2/groups -d '{
  "id": "my-group",
  "apps": [
    {
      "id": "my-app", "dependencies": ["my-other-app"] ... },
    {
      "id": "my-other-app", "dependencies": ["yet-another-app"] ... },
    {
      "id": "yet-another-app", ... }
  ]
}'
Deploying dependent groups with Marathon

```json
{
  "steps": [
    {
      "actions": [{
        "action": "StartApplication",
        "app": "/yet-another-app"
      }]
    },
    {
      "actions": [{
        "action": "StartApplication",
        "app": "/my-other-app"
      }]
    },
    {
      "actions": [{
        "action": "StartApplication",
        "app": "/my-app"
      }]
    }
  ]
}
```
Deploying dependent groups with Marathon
Deploying dependent groups with Marathon

A → B → C

D → E

= depends on
Deploying dependent groups with Marathon

Layered deployment

1. A → B → C

2. D → E
Can we do better?
We can do better!

→ = notifies

A →D
B →D
C →E
Writing custom frameworks
Bindings for many languages

- C++
- Java
- Scala
- Clojure
- Haskell
- Go
- Python
## Simple API

- registered
- reregistered
- resourceOffers
- offerRescinded
- statusUpdate
- frameworkMessage
- disconnected
- slaveLost
- executorLost
- error
akka-mesos (work in progress)

- Non-blocking
- Stream of messages instead of callbacks
- Wrappers around the protobuf messages
- No JNI
Auto scaling applications
Auto scaling applications
Auto scaling applications

```scala
system.actorOf(ClusterSingletonManager.props(
  singletonProps = Props(classOf[MesosScheduler]),
  singletonName = "consumer",
  terminationMessage = PoisonPill,
  role = Some("master"),
  name = "mesos-scheduler")
```
Auto scaling applications

```scala
val frameworkInfo = FrameworkInfo(
  name = "autoscale",
  user = "user",
  failoverTimeout = Some(5.minutes),
  checkpoint = Some(true)
)
```
Auto scaling applications

```scala
framework = Mesos(context.system)
  .registerFramework(
    Success(PID("127.0.0.1", 5050, "master")),
    frameworkInfo)

implicit val materializer = ActorFlowMaterializer()

framework.schedulerMessages.runForeach (self ! _)
Cluster(context.system).subscribe(self, classOf[ClusterMetricsChanged])
```
Auto scaling applications

case ClusterMetricsChanged(metrics) =>
  val (loadSum, heapSum) = metrics.foldLeft((0.0, 0.0)) {
    case ((loadAcc, heapAcc), metric) =>
      val load = metric.metrics.collectFirst {
        case x if x.name == "system-load-average" => x.value.doubleValue()
      } getOrElse 0.0
      val heap = metric.metrics.collectFirst {
        case x if x.name == "heap-memory-used" => x.value.doubleValue()
      } getOrElse 0.0
      (loadAcc + load, heapAcc + heap)
  }

  val loadAvg = loadSum / metrics.size
  val heapAvg = heapSum / metrics.size
Auto scaling applications

```java
if ((loadAvg > upperLoadThreshold || heapAvg > upperHeapThreshold) && scaleBackoff.isOverdue()) {
    log.info("Scaling up!")
    scaleUp = true
    scaleBackoff = 30.seconds.fromNow
}
```
else if ((loadAvg < lowerLoadThreshold &&
    heapAvg < lowerHeapThreshold) &&
    scaleBackoff.isOverdue() &&
    runningTasks.nonEmpty) {
    log.info("Scaling down!")
    scaleUp = false
    val task = runningTasks.head
    framework.driver.killTask(task)
    scaleBackoff = 30.seconds.fromNow
}
Auto scaling applications

```scala
case ResourceOffers(offers) if scaleUp =>
  val matchingOffer = findMatchingOffer(offers)

  matchingOffer foreach { offer =>
    val taskInfo = buildTaskInfo(offer)
    framework.driver.launchTasks(Seq(taskInfo), Seq(offer.id))
    scaleUp = false
  }

  (offers diff matchingOffer.toSeq).foreach { offer =>
    framework.driver.declineOffer(offer.id)
  }
```
Auto scaling applications

case ResourceOffers(offers) =>
  log.info("No need to scale, declining offers.")
  offers.foreach(offer => framework.driver.declineOffer(offer.id))
case StatusUpdate(update) =>
  update.status.state match {
    case TaskRunning =>
      runningTasks += update.status.taskId
    case TaskKilled | TaskError | TaskFailed | TaskFinished =>
      runningTasks -= update.status.taskId
    case _ =>
  }
  log.info(s"${update.status.taskId} is now ${update.status.state}")
framework.driver.acknowledgeStatusUpdate(update)
Check out the projects

https://github.com/mesosphere/marathon

https://github.com/drexin/akka-mesos

https://github.com/apache/mesos
Launch a Mesosphere cluster on Google Compute Engine

https://google.mesosphere.com
Join the community

http://mesos.apache.org

https://mesosphere.github.io/marathon/
We are hiring!

http://mesosphere.com/jobs
Thanks for your attention